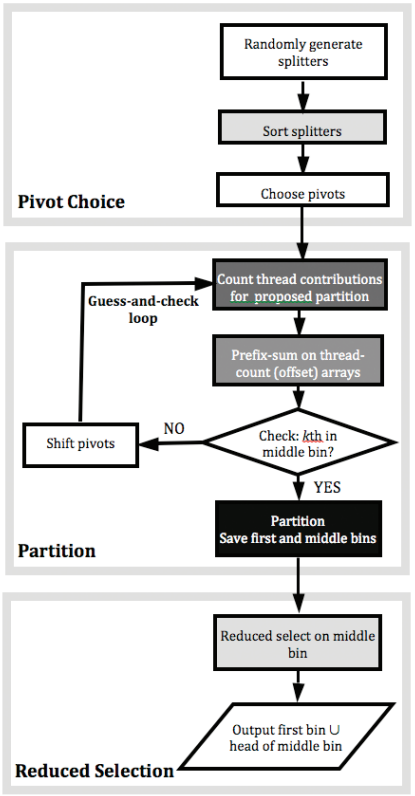
Randomized Selection on the GPU

Laura M. Monroe, HPC-5; Joanne R. Wendelberger, Sarah E. Michalak, CCS-6

We discuss the implementation of a fast, memory-sparing probabilistic top ***k*** selection algorithm on the graphics processing unit (GPU) [1]. This probabilistic approach permits exploitation of the massive parallelism of the GPU. This algorithm is linear in list length and is much faster than previously known GPU selections.

Fig. 1. Flowchart representing the selection algorithm. Those kernels making heavy use of the GPU are shaded, with the darkest being the most time-consuming.



Selection of the top k elements from a strictly weakly ordered set is one of the classic computer science algorithms [2]. Selection has application to a wide variety of statistical, computational, and database-processing problems, such as computer vision, robotics, data mining and image processing. Our initial motivation was in support of a graphics processing unit (GPU) implementation of the CLEAN algorithm, used in radioastronomy to remove noise from images that are generated from multiple antennas [3,4].

Las Vegas algorithms like this selection are a form of stochastic optimization, and optimize time spent and memory used. They can be well suited to more general parallel processors with limited amounts of fast memory. For problems having a quantifiable and efficiently checkable randomized way to guess an answer, the Las Vegas approach may be much more efficient than a direct calculation.

Selection of GPU. The GPU is an accelerator that is seeing increasing use in high-performance computing. It has a massively parallel single instruction, multiple data (SIMD) architecture and a hierarchical memory structure. This architecture is best suited to parallel algorithms that make use of the many processors without demanding much inter-processor data movement. Advantages of the GPU include massive parallelism providing much performance improvement (for parallel algorithms). Disadvantages include the small amount of fast memory close to each processor, necessitating careful and limited data placement, and the constrained bandwidth from the central processing unit (CPU) to GPU.

The naïve approaches to selection do not map well to the GPU. These compare elements on a global basis, so they call for an ample amount of fast memory to hold elements for comparison, or else a large amount of data movement across memory hierarchies. Selection via sort is another approach: after a sort, the top k elements are trivially extracted. Although fast sorts exist on the GPU, this method does more work than needed, and therefore is wasteful. If the kth element were only known ahead of time, selection would be well-suited to the GPU—elements could be read into the processors, compared against the known kth key, and saved if their rank is higher than the kth. This process is highly parallel and requires little inter-processor data movement. Unfortunately, neither the kth element nor the list distribution is known before execution.

It is possible, however, to guess the kth element probabilistically, and thus proceed as if the kth were known. Two list elements are found between which the kth element is located with arbitrary probability. This probabilistic guess eliminates the need for global data availability and transforms the problem into the highly parallel version described above. Although this algorithm is probabilistic, it always calculates a correct, unordered set of the top k elements, always terminates, and terminates after only one iteration with arbitrarily chosen probability.

This randomized selection is an example of a Las Vegas algorithm [5]. A Las Vegas algorithm is one of a class of algorithms for which the methods are probabilistic, but the result is always correct. Similar selections were developed by Motwani and Raghavan [6] and by Bader [7], but these methods used a different method of guessing the kth element.

For more information contact Laura Monroe at lmonroe@lanl.gov

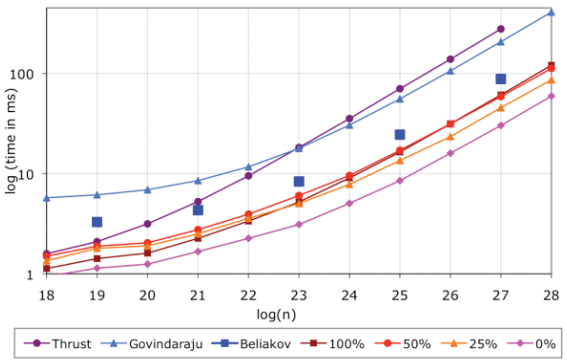


Fig. 2. Timings for different k/n, where k is the rank of the element selected for and n is the list length. Each line marked by a percentage represents a different value for k/n. These are compared to three other methods of GPU selection.

Randomized Selection Algorithm. The algorithm proceeds via an iterative, probabilistic guess-and-check process for a three-way partition. The algorithm is composed of three parts: 1) the probabilistic guess of two pivots between which the *k*th element is likely to fall, 2) partition based on the pivots, and 3) reduced selection on the smallest (middle) part of the partition, holding the *k*th element. Both keys and values are extracted. The original list is untouched.

To guess the two pivots, we use a set of randomly chosen keys (called splitters) from the list. We sort the splitters and use them to conceptually partition the list, thus "flattening out" the distribution. This conceptual partition produces buckets that typically contain roughly the same number of keys, so one can reason probabilistically about them. We use this conceptual structure to estimate the probability of the *k*th key falling into each bucket.

This estimate makes use of a binomial approximation to determine the probability that the *k*th key is in a specific bucket or set of buckets. If the *k*th key lies outside the range of the splitters, the approximation may be poor for the end buckets, but the impact will be minimal if the tail probabilities associated with the end buckets are small. Numerical studies confirm that the method performs well. The bucket probabilities are added and a concatenation of buckets is chosen so that the concatenation 1) contains the *k*th key with desired probability, and 2) is relatively small in size. The two splitters bounding the concatenation then serve as pivots.

Keys and associated values are partitioned into bins defined by the pivots and counted. If the count shows that the *k*th key is in the bin bounded by the pivots, the algorithm completes with a selection on this relatively small bin. Otherwise, new pivots are chosen and the process iterates.

Performance on the GPU. This algorithm makes excellent use of the parallelism available on the GPU. Our largest runs used 64 million threads, delivering good thread-level parallelism. The algorithm is not particularly compute-intensive, but truly benefits from the massive parallelism of the GPU. The latency from the main memory to the GPU across the PCIe bus can be many times that of selection itself, so if the overall calculation is on the GPU, it is far better to efficiently select there as well. The limiting factor on the maximum size of the original list is the amount of global memory on the GPU.

We tested on the NVIDIA Quadros 5000 and 6000, and GeForce GTX 285 graphics cards. We show results from runs on the 6000. We compared randomized selection, selection-via-sort using Thrust radix sort [8], construction of the *k*th element [9], and construction by minimization of a convex function [10]. We experimented on lists of 32-bit integer keys, of length from 2^{18} to 2^{29}. These were ordered randomly, randomly with high entropy, sorted and sorted backwards, and also included a real 32-bit 2^{18}-pixel image from radio astronomy. Our results show: 1) speedups for larger lists are three to six times faster than selections via sort and direct construction, and one to two times faster than selections via minimization of a convex function; and 2) list sizes can be processed that are up to four times longer than those possible using sort.

[1] Monroe, L. et al., "Randomized Selection on the GPU," *Proc ACM SIGGRAPH Symposium High Perf Graphics 2011* (2011).
[2] Knuth, D., *Sorting by Selection. The Art of Computer Programming,* Vol, **3,** Third Edition, Addison-Wesley (1997).
[3] Högbom, J.A., *Astron Astrophys Suppl* **15,** 417 (1974).
[4] Clark, B.G., *Astron Astrophys* **89,** 377 (1980).
[5] Babai, L., "Monte Carlo Algorithms in Graph Isomorphism Testing," Tech. Rep. DMS 79-10, Universite de Montreal (1979).
[6] Motwani, R. and P. Raghavan, *Randomized Algorithms,* Cambridge University Press (1995).
[7] Bader, D.A., *J Parallel Distr Comput* **64,** 1051 (2004).
[8] Hoberock, J. and N. Bell, http://code.google.com/p/thrust/2010 (2010).
[9] Govindaraju, N. et al., "Fast Computation of Database Operations Using Graphics Processors," *Proc ACM SIGMOD* (2004).
[10] Beliakov, G., arXiv:1104.2732v1 (2011).

Funding Acknowledgments
LANL Laboratory Directed Research and Development Program